Curso Microcontroladores Atmel AVR

Tecnologico de Morelia
Departamento Electronica
talfaro2000@yahoo.com

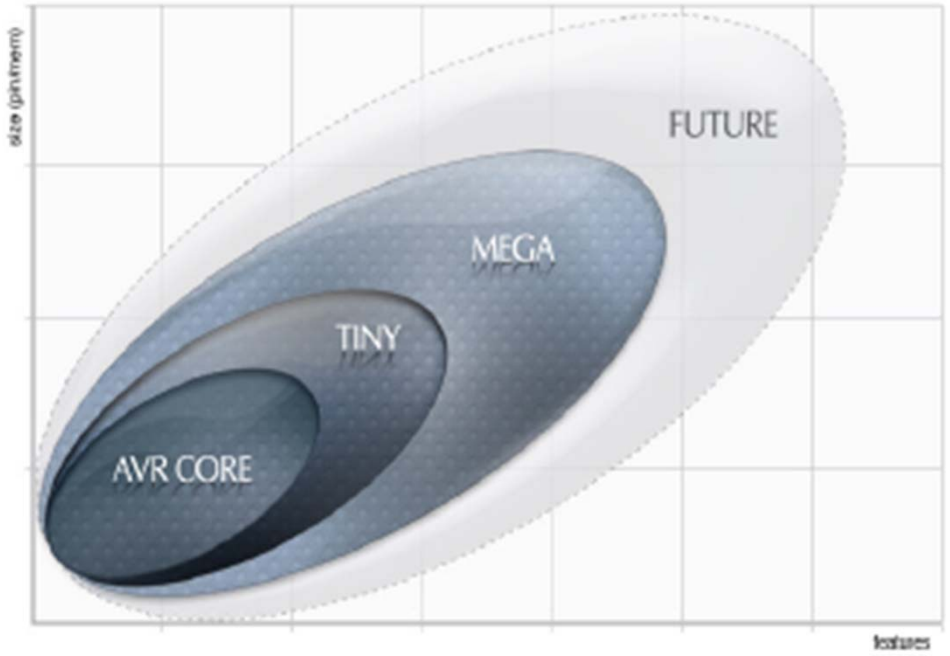
Familia AVR

TINY AVR
- 8 - 32 pin, microcontroladores de proposito general
- 16 miembros de la familia

MEGA AVR
- 32 - 100 pin, microcontroladores de proposito general
- 23 miembros de la familia

ASSP AVRs
- USB, CAN y LCD
- Control Motor e iluminacion
- Automotriz
- Administracion de Baterias
- 8 miembros de la familia

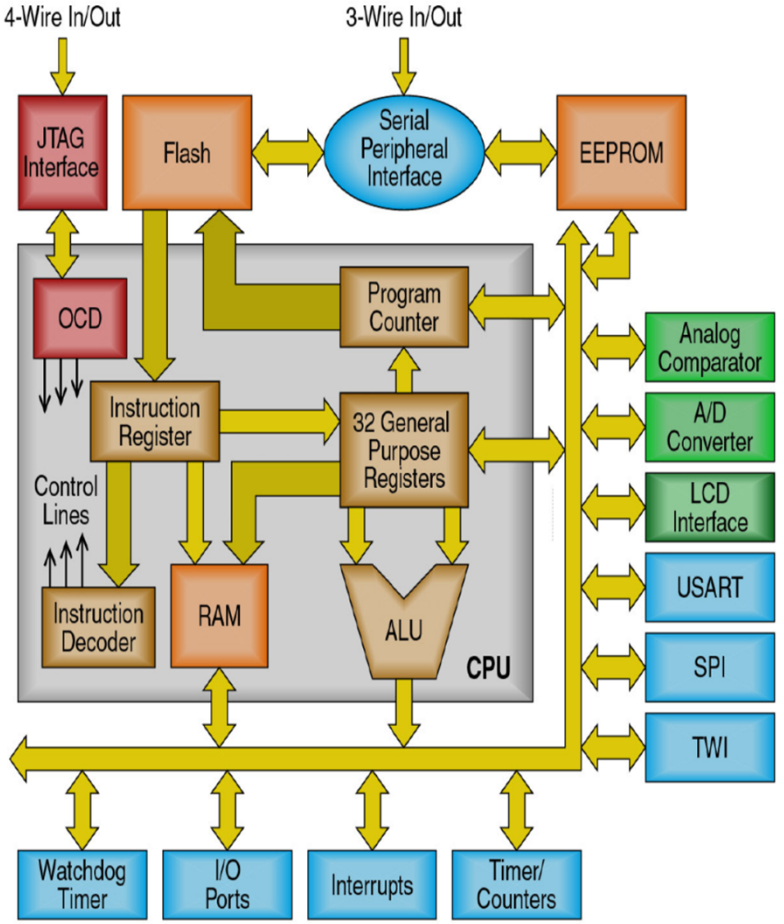
-Dispositivos con rango desde 1 a 256KB
-Rango del numero de terminales desde 8 a 100
-Compatibilidad total de codigo
-Caracteristica/Pin compatibles entre familias
-Un conjunto de herramientas de desarrollo

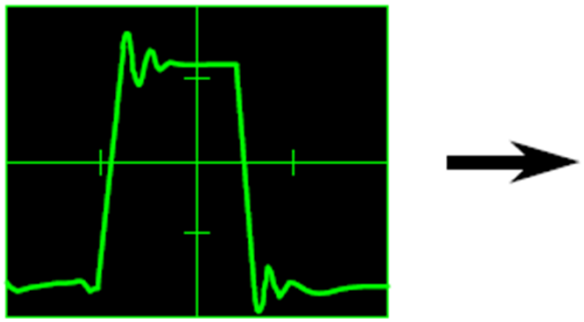


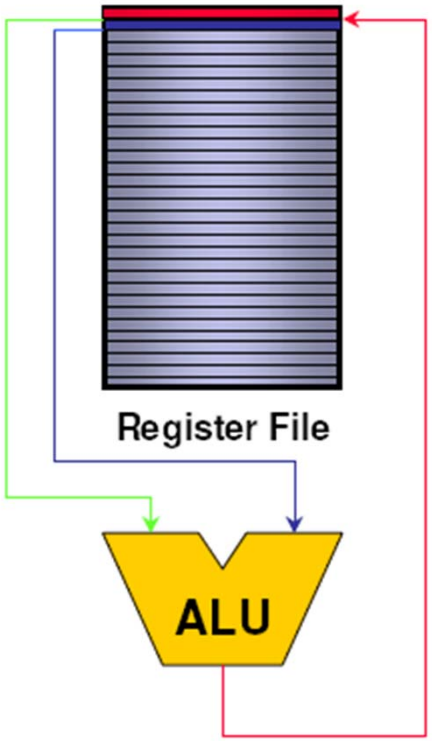
Arquitectura AVR

Arquitectura RISC con conjunto de instrucciones CISC

- Conjunto de Instrucciones poderoso para C y ensamblador
- Escalables
- Mismo Nucleo poderoso en todos los dispositivos AVR
- Execucion en un solo ciclo
- Una instruccion por reloj externo
- Consumo de potencia baja
- 32 registros de trabajo
- Todos conectados directamente a la ALU!
- Nucleo (core) muy eficiente
- 20 MIPS @ 20MHz
- Alto nivel de integracion
- Costo del sistema total bajo







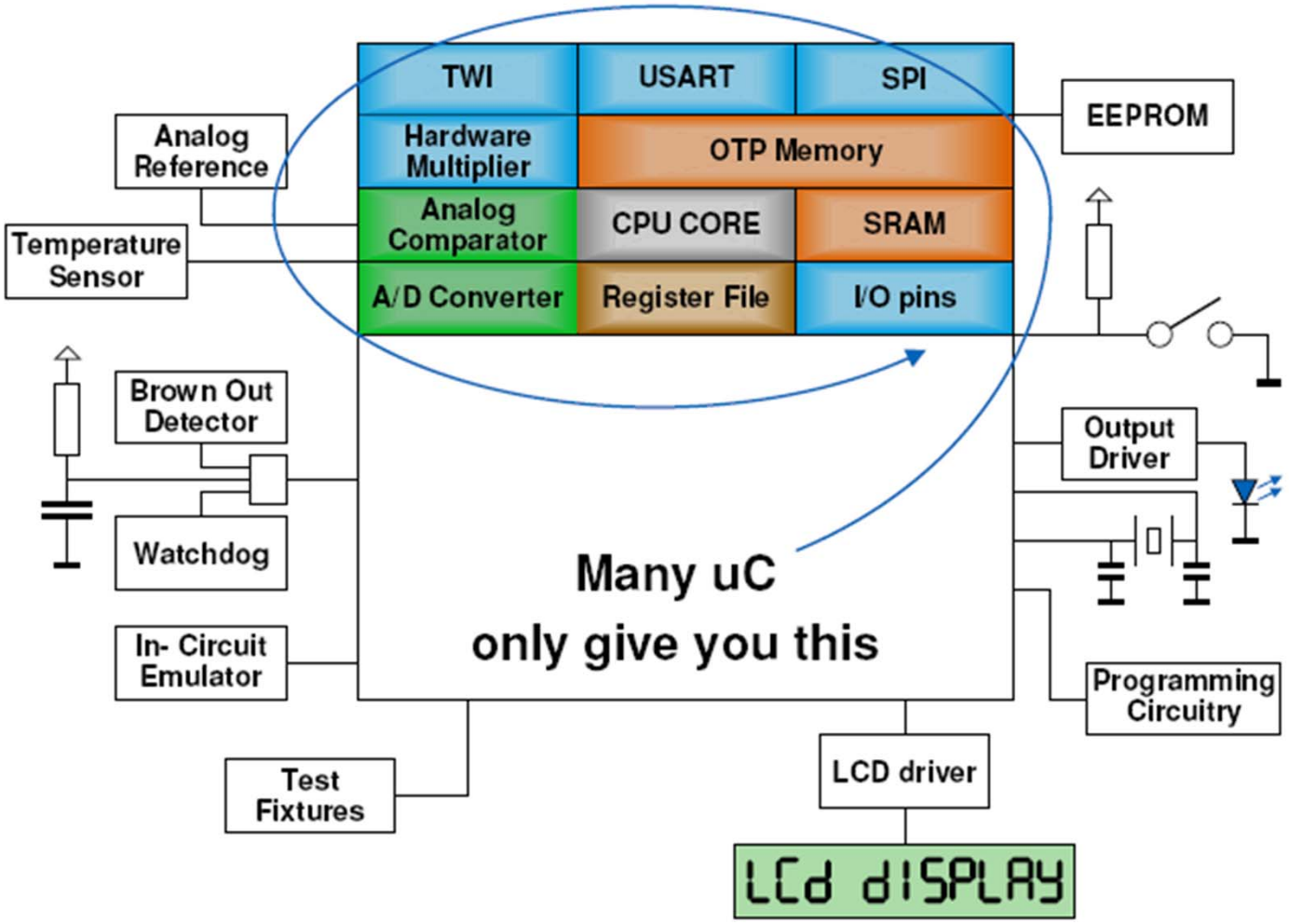
Las operaciones con registros
Toman un SOLO pulso de reloj
En la entrada externa de reloj

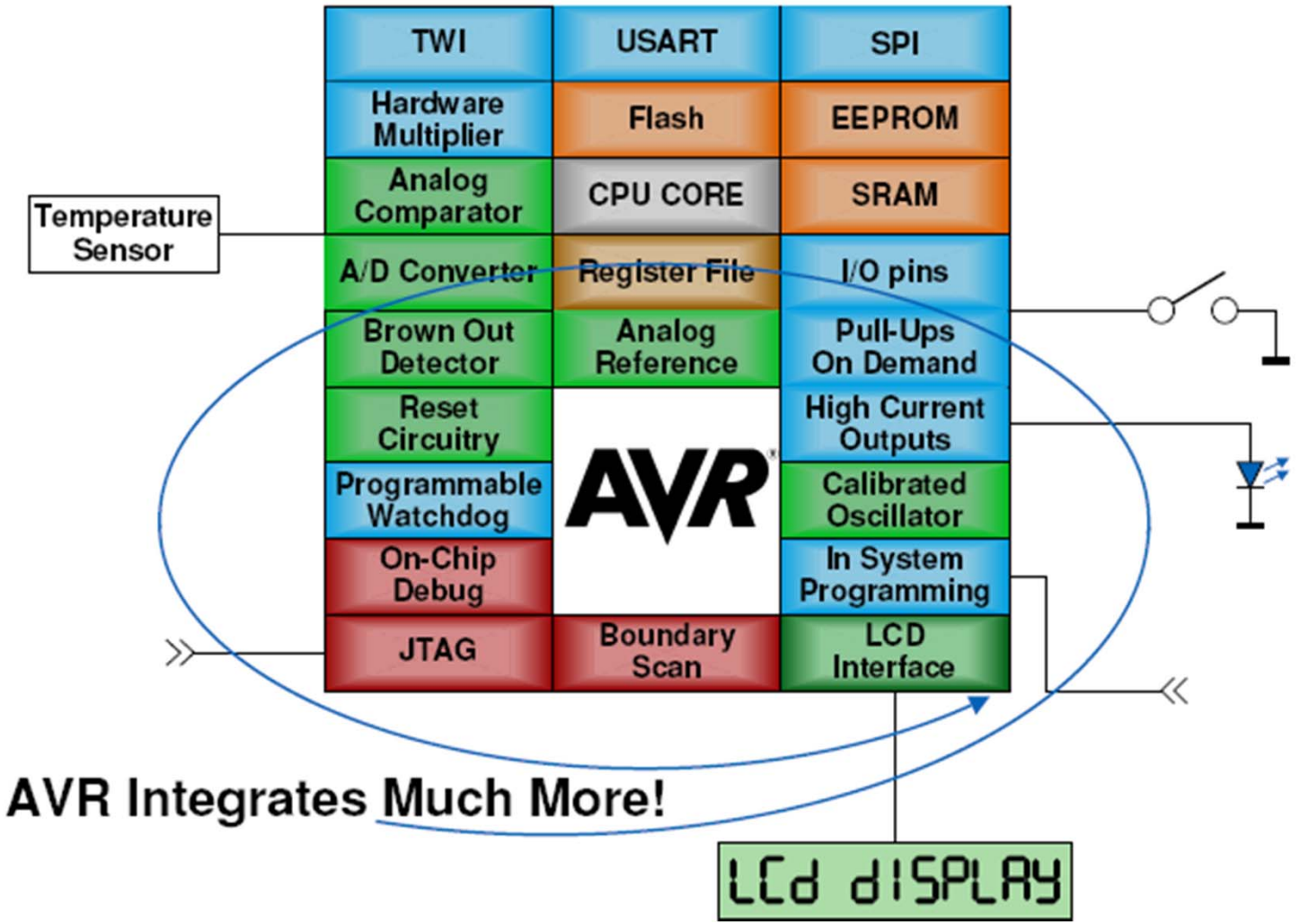
20MIPS @ 20MHz

Tamano Codigo y tiempo de ejecucion

Device	Max Speed [MHz]	Code Size [Bytes]	Cycles	Execution Time [uS]
ATmega16	16	32	227	14.2
MSP430	8	34	246	30.8
T89C51RD2	20	57	4200	210.0
PIC18F452	40	92	716	17.9
PIC16C74	20	87	2492	124.6
68HC11	12	59	1238	103.2

Solucion en un solo circuito integrado




TWI
USART
SPI
Hardware Multiplier
Flash
EEPROM
Analog Comparator
CPU CORE
SRAM
Temperature Sensor
A/D Converter
Register File
I/O pins
Brown Out Detector
Analog Reference
Pull-Ups On Demand
Reset Circuitry
AVR
High Current Outputs
Programmable Watchdog
Calibrated Oscillator
On-Chip Debug
In System Programming
JTAG
Boundary Scan
LCD Interface
AVR Integrates Much More!
LCd dISPLAY

ATMEGA48

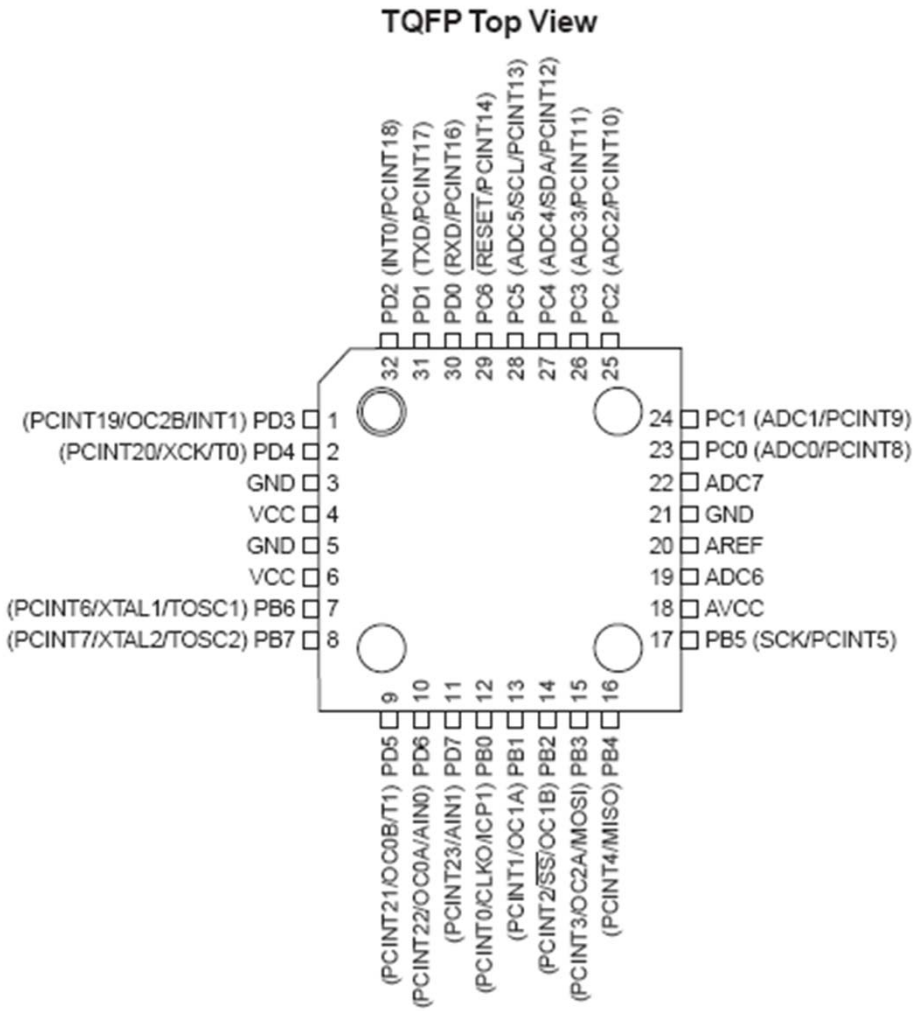
Caracteristicas

Features

- High Performance, Low Power AVR® 8-Bit Microcontroller
- Advanced RISC Architecture
  - 131 Powerful Instructions – Most Single Clock Cycle Execution
  - 32 x 8 General Purpose Working Registers
  - Fully Static Operation
  - Up to 20 MIPS Throughput at 20 MHz
  - On-chip 2-cycle Multiplier
- High Endurance Non-volatile Memory segments
  - 4/8/16K Bytes of In-System Self-programmable Flash program memory
  - 256/512/512 Bytes EEPROM
  - 512/1K/1K Bytes Internal SRAM
  - Write/Erase cyles: 10,000 Flash/100,000 EEPROM (1)(3)
  - Data retention: 20 years at 85°C/100 years at 25°C (2)(3)
  - Optional Boot Code Section with Independent Lock Bits
    - In-System Programming by On-chip Boot Program
    - True Read-While-Write Operation
  - Programming Lock for Software Security
- Peripheral Features
  - Two 8-bit Timer/Counters with Separate Prescaler and Compare Mode
  - One 16-bit Timer/Counter with Separate Prescaler, Compare Mode, and Capture Mode
  - Real Time Counter with Separate Oscillator
  - Six PWM Channels
  - 8-channel 10-bit ADC in TQFP and QFN/MLF package
  - 6-channel 10-bit ADC in PDIP Package
  - Programmable Serial USART
  - Master/Slave SPI Serial Interface
  - Byte-oriented 2-wire Serial Interface (Philips I^2C compatible)
  - Programmable Watchdog Timer with Separate On-chip Oscillator
  - On-chip Analog Comparator
  - Interrupt and Wake-up on Pin Change

- Special Microcontroller Features
  - Power-on Reset and Programmable Brown-out Detection
  - Internal Calibrated Oscillator
  - External and Internal Interrupt Sources
  - Five Sleep Modes: Idle, ADC Noise Reduction, Power-save, Power-down, and Standby
- I/O and Packages
  - 23 Programmable I/O Lines
  - 28-pin PDIP, 32-lead TQFP, 28-pad QFN/MLF and 32-pad QFN/MLF
- Operating Voltage:
  - 1.8 - 5.5V for ATmega48V/88V/168V
  - 2.7 - 5.5V for ATmega48/88/168
- Temperature Range:
  - -40°C to 85°C
- Speed Grade:
  - ATmega48V/88V/168V: 0 - 4 MHz @ 1.8 - 5.5V, 0 - 10 MHz @ 2.7 - 5.5V
  - ATmega48/88/168: 0 - 10 MHz @ 2.7 - 5.5V, 0 - 20 MHz @ 4.5 - 5.5V
- Low Power Consumption
  - Active Mode:
    250 µA at 1 MHz, 1.8V
    15 µA at 32 kHz, 1.8V (including Oscillator)
  - Power-down Mode:
    0.1µA at 1.8V

Encapsulados



PDIP

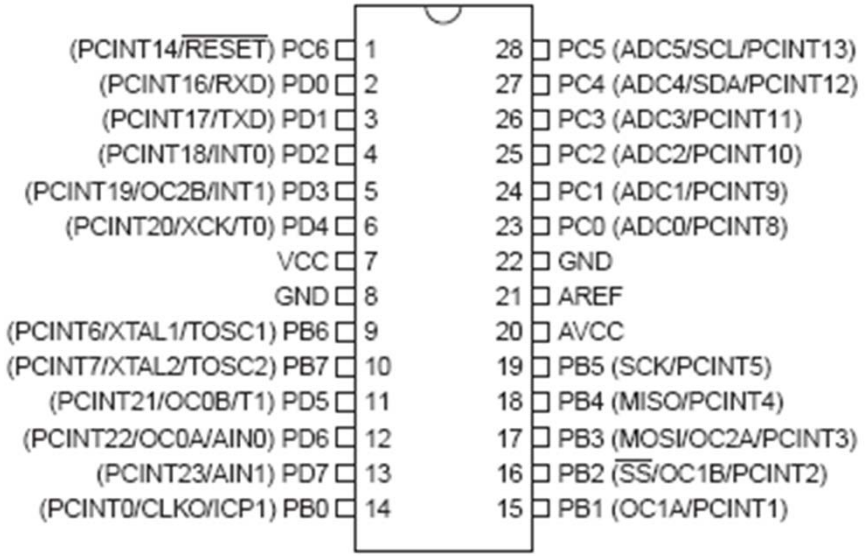
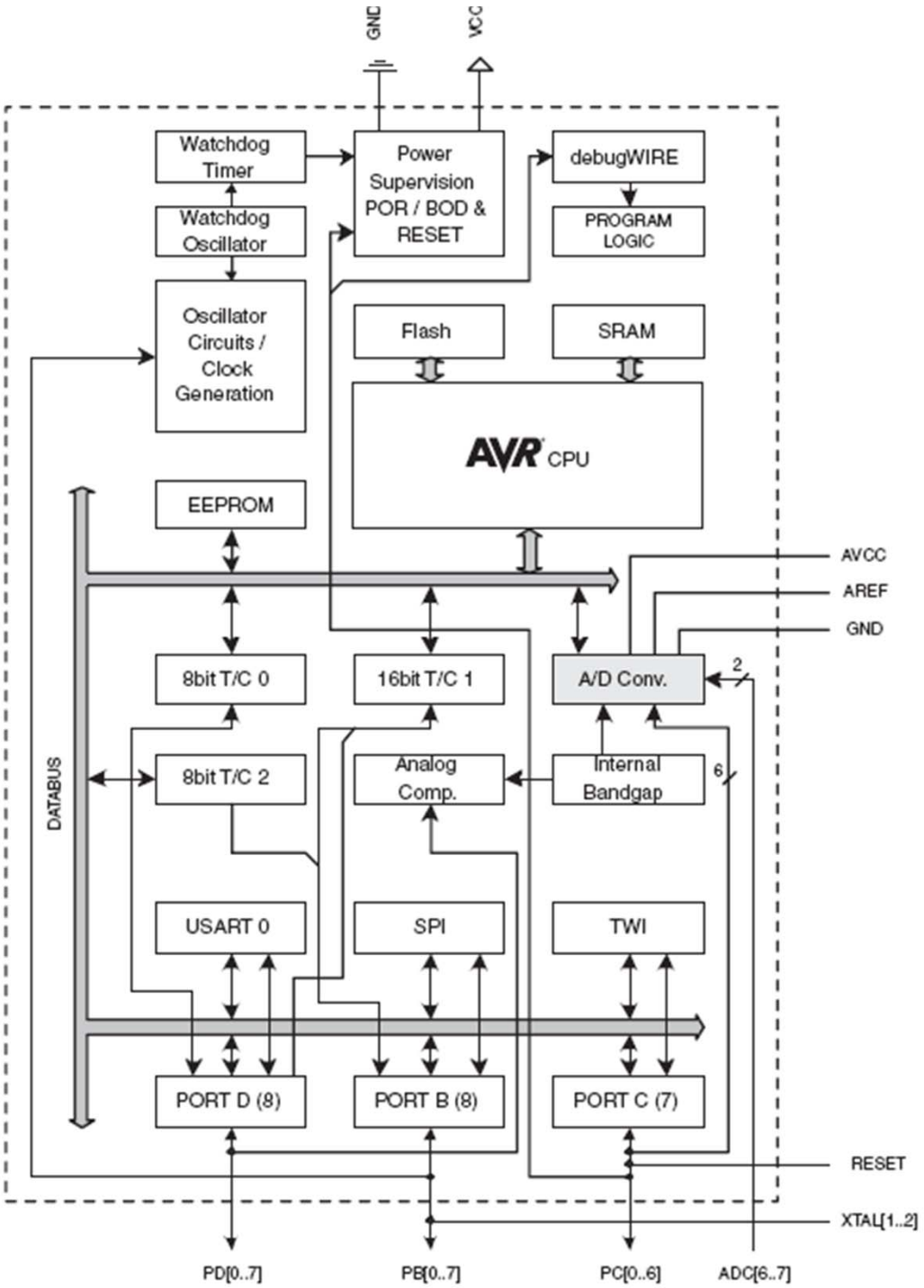


Diagrama
de Bloques



Variantes

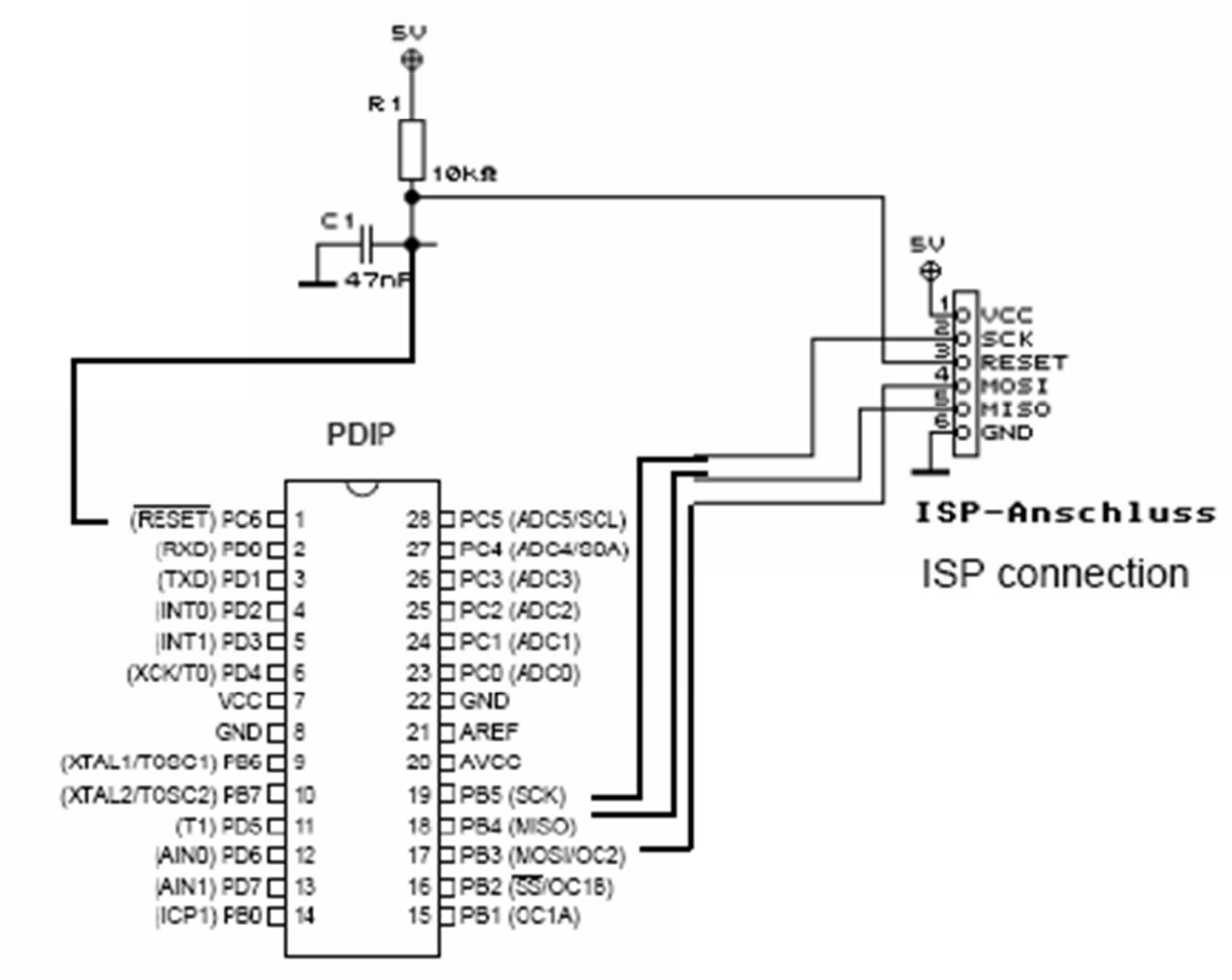
Table 2-1. Memory Size Summary

Device	Flash	EEPROM	RAM	Interrupt Vector Size
ATmega48	4K Bytes	256 Bytes	512 Bytes	1 instruction word/vector
ATmega88	8K Bytes	512 Bytes	1K Bytes	1 instruction word/vector
ATmega168	16K Bytes	512 Bytes	1K Bytes	2 instruction words/vector

PROGRAMACION DEL MICROCONTROLADOR

In system Programming

Circuit



Programacion usando AVRDUDE

En una ventana de MSDOS introducir la siguiente linea de comando:

avrdude –p códigodemicro –c usbasp –e –u –U flash:w:nombrearchivo.hex

Donde “codigodemicro” es una identificacion para el microcontrolador
Que se quiere programar.

“nombrearchivo.hex” es el archivo (generado por winnAVR o Codevision
u otro compilador) que se quiere programar en el microcontrolador.

“usbasp” es el nombre del programador.

“flash” y “w” es para indicarle que se quiere escribir (w) en la memoria flash.

Para mas informacion consultar el manual de AVRDUDE.

Codigos de los Micros

Microcontrolador	códigodemicro
AT90CAN128	c128
AT90PWM2	pwm2
AT90PWM3	pwm3
AT90S1200	1200
AT90S2313	2313
AT90S2333	2333
AT90S2343 (*)	2343
AT90S4414	4414
AT90S4433	4433
AT90S4434	4434
AT90S8515	8515
AT90S8535	8535
AT90USB646	usb646x
AT90USB647	usb647x
AT90USB1286	usb1286x
AT90USB1287	usb1287x
ATmega103	m103
ATmega128	m128
ATmega1280	m1280
ATmega1281	m1281
ATmega16	m16
ATmega161	m161
ATmega162	m162
ATmega163	m163
ATmega164	m164

ATmega168	`m168`
ATmega169	`m169`
ATmega2560 (**)	`m2560`
ATmega2561 (**)	`m2561`
ATmega32	`m32`
ATmega324	`m324`
Atmega325	`m325x`
ATmega329	`m329`
ATmega3250	`m3250x`
ATmega3290	`m3290`
ATmega48	`m48`
ATmega64	`m64`
ATmega640	`m640`
ATmega645	`m645x`
ATmega644	`m644`
ATmega649	`m649`
ATmega6450	`m6450x`
ATmega6490	`m6490`
ATmega8	`m8`
ATmega8515	`m8515`
ATmega8535	`m8535`

ATmega88	`m88`
ATtiny11	`t11`
ATtiny12	`t12`
ATtiny13	`t13`
ATtiny15	`t15`
ATtiny2313	`t2313`
ATtiny24	`t24x`
ATtiny25	`t25`
ATtiny26	`t26`
ATtiny261	`t61`
ATtiny44	`t44x`
ATtiny45	`t45`
ATtiny461	`t461`
ATtiny84	`t84x`
ATtiny85	`t85`
ATtiny861	`t861`

LENGUAJE C

Herramientas de Programacion

- Editor
- Compilador (Compiler)
- Enlazador (Linker)
- Cargador (Loader)
- Depurador (Debugger)

- Ambiente de Desarrollo Integrado (IDE)
  - Interface para los comandos basicos

WinAVR

- WinAVR es una suite de herramientas ejecutables, (open source software) para el desarrrollo de programacion de microprocesadores Atmel AVR por medio de la plataforma Windows. Incluye un compilador GNU GCC para C y C++.
  - http://winavr.sourceforge.net/

Comentarios en C

- /* Multi
- Linea
- Comentarios */
- //comentarios al final de linea

Includes

- Incluye un archivo – tipicamente un archivo de cabecera (header file) conteniendo declaraciones de simbolos.
- #include <avr/io.h>
  - Este es un archivo estandar del sistema
    - El compilador estandard sabe en que carpetas buscar este archivo
- #include "mystuff.h"
  - Este es un archivo especifico del proyecto
    - El compilador busca en la carpeta del proyecto este archivo

Definiciones

- #define symbol *lo que se desea*
- Este es llamado un macro
  - define un simbolo como una cadena de texto
- Cada ocurrencia del simbolo es reemplazado por la cadena despues del simbolo
- Esto es hecho por el preprocesador de C

Variables

- Especifican un tipo, ambiente, tiempo de vida, y nombre
  - Tipo es char, int, float, etc
    - En AVR tipos especiales como uint8_t, int16_t, etc
  - Ambiente es
    - Local (block) – declaradas dentro de funciones
    - Global (file) – declaradas fuera de funciones
    - Program (external) – Usedas como enlaces entre varios archivos
  - Tiempo de vida es
    - Automatico (default para variables locales) (en la pila)
    - Estatico (default para variables globales y de programa) (en ram)
    - Dinamico (localidad de almacenaje controlado por programa) (en heap)

Declaracion de Variables

Las declaraciones de las variables se hacen de la siguiente forma:
Con signo o sin signo tipo nombre; //Comentarios

Ejemplos
unsigned char x,y,temperatura;
unsigned int var1; *// Es una variable entera sin signo que llamamos var1*

Tipos de Variables

Tipo	Tamaño en bits	Rango
Char	8	127
Unsigned char	8	255
signed char	8	127
Int	16	32767
short int	16	32767
Unsigned int	16	65535
signed int	16	32767
long int	32	2147483647
Unsigned long int	32	4294967295
signed long int	32	2147483647
Float	32	±1.75 e ±3.402 e 38
Double	32	±1.75 e ±3.402 e 38

Tipos de Datos

Variable o constante	Formato
Decimal	Número
Hexadecimal	0x número hexadecimal
Binario	0b número binario
Octal	0 número octal
Caracter	‘a’
Cadena	“ esta es una cadena”

```
x=20;          // x es 20 decimal
x=0x14;        //x es 14 hexadecimal que convertido a decimal es 20 decimal
x=0b00010100;  //x se manejado en binario que es equivalente a 20 decimal
x=024;         //x se maneja en octal que es 20 decimal
```

Arreglos

Un arreglo es un conjunto de datos que pueden ser accesado a través de un índice.
Su declaración se hace así:

nombre del arreglo [número de elementos]={elemento1, elemento2,.. elemento n }

Ejemplo
char arreglo1 [3]={0x0a,0x38,0x23};

El primer elemento es el 0, en la declaración anterior del arreglo éste se definió de tres elementos, siendo el primer elemento el 0 y el último elemento el número 2, vea el siguiente ejemplo:

x=arreglo1[0]; // x vale 0x0a ya que accesó al primer elemento del arreglo

Arreglos Multidimensionales

Se puede declarar un arreglo de dos dimensiones que se interpretaría como fila y columna ejemplo:

```
char arreglo_multi_dim  [2,3]= {1,2,3}, {,4,5,6};      // Es un arreglo de dos
                                                        //filas y tres columnas
```

Operadores aritmeticos

Símbolo	Operación
+	Suma
-	Resta
*	Multiplicación
/	División
%	División Módulo, y el resultado es el residuo

Operadores para el manejo de bits

Símbolo	Descripción
&	And Bit a Bit
|	OR bit a Bit
^	Or exclusivo Bit a Bit
<<	Corrimiento a la Izquierda
>>	Corrimiento a la derecha
~	Complemento a unos (inversión de bits)

Operadores de relacion

Operador	Descripción
>	Mayor que
>=	Mayor o igual que
<	Menor que
<=	Menor Igual que
==	Igual que
!=	Distinto de
&&	Y también si
||	O si

Estructura if - else

La estructura *if else* es: si se cumple el *if* se hace lo que está abajo del *if* y sino se cumple se hace lo que está debajo del *else*

```
if (Var1==10)
{
        Var1=0;
        Var2=20;
        x=14;
}
else
{
        Var1=++;
        x=20;
}
```

Estructura else - if

Donde se cumpla la condición rompe las siguientes comparaciones, no evalúa todas, pero si se cumple hasta abajo habrá evaluado todas las comparaciones.

```
if (Var1==10)
        Var1=0;
else if (Var1==09)
        Var1=1;
else if (Var1==08)
        Var1=2;
else
        Var1=3;
```

Estructura while

El *while* evalúa lo que hay entre los paréntesis y mientras se cumpla ejecuta lo que está debajo del *while*.

```
while ((porta==0x01) &&(var1==0))
{
        portb++ ;          //Si porta=0 y Var1=0, se incrementa en uno el portb
}
```

Estructura do -while

A diferencia del *while*, esta instrucción ejecuta por lo menos una vez lo que está abajo del *do*, después evalúa el *while* y si es válido ejecuta nuevamente lo que está abajo del *do*.

```
do
{
        Portb++ ;          //Si porta=0 y Var1=0, se incrementa en uno el portb
}
While ((porta==0x01) &&(Var1==0))
```

Estructura for

La estructura del *for* es (inicio; condición de paro; incremento del índice)

```
for (i=0;i<=10000,i++)
{
        var1++;  //i tendría que ser declarado como  int para que cumpla con el rango
}
```

Ruptura de Ciclos con Break

Se puede terminar un ciclo *for, while y do while* mediante el *break* sin necesidad de terminar las iteraciones o de que se vuelva falsa la condición.

```
for (i=0;i<=10000,i++)
{
var1++;
if (var2==0)  //Si var2=0 se rompe ciclo, quedando el i en donde se haya cumplido el var2
break;
}
var3=i;
```

Ciclos infinitos

```
for(;;)
{
        Se ejecuta infinitamente las instrucciones aquí colocadas
}
```

Otra forma de generar ciclos infinitos:

```
while(1)
{
        Se ejecuta infinitamente las instrucciones aquí colocadas
}
```

Prototipos de funciones (encabezados)

- Las funciones deben serer declaradas antes de su uso
  - Una declaracion es solamente el encabezado de la funcion no el cuerpo
    - Las declaraciones de funciones (encabezados) normalmente son colocadas en archivos include
  - Una definicion de una funcion incluye el cuerpo de la funcion
- void function_name(int, float); //encabezado
  - Se requiere tipo de dato de retorno; void significa que nada es regresado
  - Tipos de los argumentos

Main

- Cada programa en C nesecita una funcion llamada *main*

```
int main(void){
    int x; //local var
    x = add(3, 9); //llamada a una funcion
    return 0;
}
```

- El cuerpo del programa va dentro de los braques
- Esta funcion normalmente regresa 0 como codigo que significa que todo esta bien

Otras funciones

- Las definiciones de otras funciones son colocadas debajo de main (aunque se pueden poner arriba) o en un archivo separado

```
int add(int a, int b){
  return a+b;
}
```

Loas argumentos son pasados por valor a los parametros

Operaciones con Bits

- &, |, ^, >>, <<, ~
- & - bitwise AND

```
DDRB = 0b11110000;
DDRB &= 0b00010000;
=> DDRB = 00010000
```

- | - bitwise Inclusive OR

```
DDRB = 0b00000000;
DDRB |= 0b00010000;
=> DDRB = 00010000
```

Tip: Use this (|=) if you want to keep all bits the same, but ensure certain bits are enabled (1)

- ^ - bit Exclusive OR

```
DDRB = 0b00010001;
DDRB |= 0b00010000;
=> DDRB = 00000001
```

Tip: Use ^= to toggle bits ON/OFF

- << - Left shift

  DDRD = 0b10000000;

  DDRD |= (1<<5);

  => DDRD = 10010000

- ~ - One's complement

  DDRD = 0b11111111;

  DDRD &= ~(1<<5)

  =>DDRD = 11101111

Good for switching off bits

- These operations are not only useful
  - Can make your code easier to read also

Interrupciones

- sei() y cli() habilitaran o deshabilitaran las interrupciones
- Para preservar el estado actual, usar
  - `uint8_t sreg = SREG;`
  - `cli();`
  - …
  - `SREG = sreg;`

Apuntadores

- Apuntadores son variables que contienen la direccion de un dato
  - `uint8_t num = 5;` // variable uint8
  - **`uint8_t * numPtr;`** //apuntador al valor uint8
- Operador de Direccion
  - `numPtr = &num;` //direccion de una variable
- Operador de Dereferencia
  - `PORTD = *numPtr;` //dato de la direccion en numPtr

Arreglos y Apuntadores

- Los nomres de los arreglos son apuntadores a constantes
  - Esto es, representan la direccion base del arreglo, pero solamente como un valor
    - `int16_t numbers[4] = {5, 7, -2, 6};`
    - `int16_t * aPtr;`
    - `aPtr = numeros;`
    - `aPtr = &numeros;`
      - Los ultimos 2 son identicos

Parametros por Referencia

- Las funciones en C usan solamente la llamada –por-valor

```
func_by_val(a);
```

- Se puede pasar y recibir la direccion del dato, dando por resultado la llamada-por-referencia

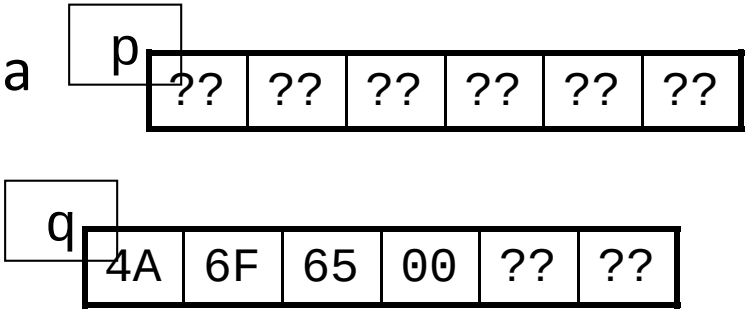
```
char a;
func_by_ref(&a);
```

```
void func(char x){
      PORTB = x;
}
```

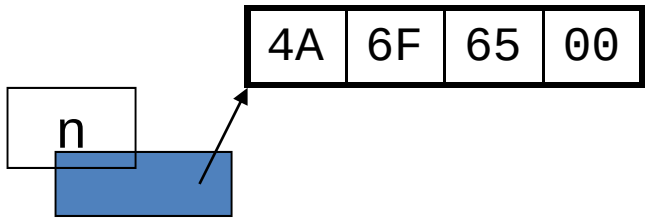
```
void func(char * x){
      *x = '$';
}
```

Cadenas

- Las cadenas son implementadas en arreglos de caracteres
  - `char p[6];` //no inicializada
  - `char q[6] = "Joe";`
    //Usa 4 de los 6 bytes



- Los arreglos pueden ser implicitamente creados y simplemente apuntados a
  - `char * n = "Joe";`



Datos en Flash

```
#include <avr/pgmspace.h>
prog_uint16_t myData[]
          = {874, 299, 14987, 37};
prog_char someText[] = "Hello World";
```

- Esto crea datos en la memoria de programa (flash)
  - Deben pertenecer a la clase de alojamiento estatico (static)
- La desventaja, es que se tiene que acceder a ellos en una forma especial
  - Hay macros y rutinas especiales de caracteres para acceder alos datos almacenados de esta forma

Program Data Macros

```
uint16_t x = pgm_read_word(myData+2);
for (i=0; i<3; i++)
   char y = pgm_read_byte(&someText[i]);
```

- Estos macros requieren la direccion del dato que sera accesado
  - El nombre de un arreglo es una direccion
    - myData+2 es lo mismo que &myData[2]
    - &someText[i] es lo mismo que someText+i

Funciones de cadenas en memoria de programa

- Agregar _P a las funciones estandars para usar las cadenas en memoria de programa
  - `int n = strlen_P(someText);`
- Las cadenas de destino deben estar en ram
  - `strcpy_P(ramStr, someText);`
- Use PGM_P para declarar un apuntador a una cadena en memoria de programa
  - `PGM_P pptr = someText;`
  - `char c;`
  - `while (c = pgm_read_byte(pptr++)){...`

Retardos (Delays)

- `#define F_CPU 4000000UL  // 4 MHz`
- `#include <avr/delay.h>`
- `_delay_us(93);`
  - Max retardo: 768 us / F_CPU en MHz
- `_delay_ms(10);`
  - Max retardo: 262.14 ms / F_CPU in MHz.
- `_delay_loop_1(24);`
  - Usa conteo de 8 bits, 3 ciclos por interacion
  - Pasa 0 para interaciones maximas (256)
- `_delay_loop_2(1039);`
  - Usa conteo de 16 bits, 4 ciclos por iteracion
  - Pasa 0 para iteraciones maximas (65536)

Numero de interacciones de lazo (para implementar el retardo) esta basado en la velocidad del procesador

math.h

- Incluye todas las funciones matematicas necesarias

  -fabs, sqrt, sin, asin, ceil, floor,

  - Notar que estos trabajan en datos punto flotante, asi que se requiere, por la mayoria de los programas en AVR, conversion de y hacia formato de enteros

Funciones de atencion a interrupciones

```
#include <avr/interrupt.h>
ISR(interrupt name){
    //El codigo de la rutina de manejo va aqui
    //No se requiere guardar y restaurar estado – es automatico
    //RETI tambien es agregada automaticamente
}
```

- Ademas de definir la rutina de manejo,se requiere habilitar/deshabilitar las interrupciones
  - `GICR |= (1<<INT0); //por ejemplo`
  - `sei();`

Proyectos Multiarchivos

- Agregar cada archivo fuente a la lista del proyecto
- Cada archivo sera compilado/ensamblado por separado
  - Use archivos de cabecera para compartir declaraciones
- Los resultados son enlazados en un solo archivo hex para ejecucion.

Dos archivos en C

```
//mainfile.c
#include <avr/io.h>
#include
  "otherfile.h"

int main(void){
  char x;
  x = tc('B');
  PORTB = x;
  return 0;
}
```

```
//otherfile.h
#ifndef otherfile_h
#define otherfile_h

char tc(char);

#endif
```

Dos Archivos en C

```
//otherfile.c
#include <avr/io.h>
#include "otherfile.h"

//toggle case
char tc(char x){
  return x ^ _BV(5);
}
```

- El archivo de cabecera es incluido donde sea declarado, y y donde sea definido
  - El compilador puede checar el uso y declaracion para consistencia

Mezclando C y Ensamblador

- Lenguaje ensamblador en linea
  - Tiene una sintasis especial para permitir acceso a las variables en C y tambien evitar problemas con resgistros usados por codigo en C.
    - asm("in %0, %1" : "=r" (localvar) : "I“ (_SFR_IO_ADDR(PIND)));
    - in r24, 0x10 ;0x10 es la direccion de PIND
    - std Y+9, r24 ;Y+9 es la direccion de localvar
- Llamando una funcion en un lenguaje desde otro lenguaje

C y Ensamblador

```
//asmfile.S
#include <avr/io.h>
;void show (uint16_t * port,
   uint8_t data)
.global show
;args:  portaddr R25:R24
;       data R22
show:
   ;tc needs its argument in R24
   push r25 ;save port address
   push r24
   mov r24, r22
   rcall tc ;toggle case
   ;r24 is case-toggled char
   pop r30 ;get port address
   pop r31
   st Z, r24   ;out!
   ret
```

- Los archivos en ensamblador requieren la extension .S
- Usa el estilo de archivos de cabecera en C
- Declara el punto de entrada(nombre de la funcion) como global de menera que el programa en C pueda “verla”

C y Ensamblador

```
//mainfile.c

#include <avr/io.h>
#include "otherfile.h"
void show(volatile uint8_t *port,
     char);

int main(void){
  char x = 'Z';
  show(&PORTB, x);
  show(&PORTC, 'a');
  return 0;
}
```

- El prototipo es agregado a main para la compilacion

Macros

-Uso de Macros

(I<<n) puede ser reemplazado por:

#define BIT(n) (I<<n);

uso: (I<<5) =>BIT(5)

-Utiles para calculos de reloj:

#define set_counter_from_seconds(t) ((CF/CKDIV I)*(t))

Esta macro permite convertir facilmente numero de ciclos de reloj a segundos.

Building

- El archivo en lenguaje ensamblador es agregado al listado de archivos del proyecto
- El comando Build ensamblara y compilara los archivos requeridos
  - Cada unidad produce un archivo objeto
  - Solamente un archivo hex es producido